contact@namiq.io · namiq.io

NamiQ

SECURITY ADVISORY

WordPress core exploited

CVE-2026-119977 · versions 4.7 to 7.1.3

Assessment & Action Plan

A race in WordPress core's own upgrade process lets an unauthenticated visitor run code on the server while an update is in progress. It affects 4.7 through 7.1.3, CISA lists it as exploited with a 6 November deadline, and the fix is 7.1.4 or your branch's backport. Who is exposed, what is at stake for the business and its data, and how to check whether someone got in first.

PREPARED BY	Hung Le, NamiQ	DATE	5 November 2026
FACTS AS OF	5 November 2026	VERSION	1.3
SEVERITY	**High: CVSS 3.1 8.1, exploited**	STATUS	Fixed in 7.1.4 and backports; exploited in the wild

NamiQ © 2026 NamiQ. All rights reserved. This document is the property of NamiQ. It may be shared unaltered, with attribution.
Companion article: namiq.io/insights/wordpress-core-cve-2026-119977

Update, 4 November 2026: added the attack sequence as a drawing and step by step, what happens after the code runs in time order, and a table of where each step can be broken. The CVE record, CISA's entry, EPSS and the fixed-release list were re-read the same day and had not changed. Nothing in the first version was withdrawn.

Update 2, 4 November 2026: added a detection chain (four events, how to join them, and how much each combination proves) with an audit rule, a process rule and the log searches to build it. It follows the attack drawing and says where it cannot see. Nothing earlier was withdrawn.

Update 3, 5 November 2026: in the detection table, a new administrator, a changed site address or an unexpected outbound connection on its own now reads as high suspicion to investigate, not as proof of compromise. Each can have an ordinary cause. Nothing else changed.

WordPress runs more of the web than any other publishing system. This flaw is in the part of WordPress that updates itself, and on 3 November CISA recorded that it is being used against real sites. The fix is out. The awkward part is that applying it is also the moment of risk.

This note is for the person who owns a WordPress site, the person who pays for one, and the agency or host that runs many. Everything is as of 4 November 2026, and every claim links to its source at the end. Where we are reasoning rather than quoting, we say so.

The short version

- CVE-2026-119977 (CWE-367, time-of-check to time-of-use race): WordPress core 4.7 through 7.1.3 uses predictable, web-accessible staging paths during full-package core upgrades. An unauthenticated attacker can run code on the server during the upgrade window. CVSS 3.1 8.1 (High), CNA Wordfence.
- CISA added it to its exploited list on 3 November with a due date of 6 November. Its coordinator rates exploitation 'active', automatable 'no' and technical impact 'total'. Ransomware use is not recorded.
- The fix is in WordPress 7.1.4, and the project has backported it to every affected branch back to 4.7. Update to the latest patch of your branch: 4.7.39, 4.8.34, 4.9.35, 5.0.31 and on, up to 6.9.11, 7.0.8 and 7.1.4.
- The same release fixes eleven other security issues, eight of them stored cross-site scripting, and updates the bundled root certificates. Taking the update is not a choice between one fix and the rest.
- Updating does not tell you whether someone came in first. The exploited list is the reason to check as well as to patch.

What the bug is

When WordPress updates itself it downloads a package, unpacks it into a working area, and then copies the new files over the old ones. That work happens on the web server, driven by WordPress's own code. The CVE record says the working paths are predictable and can be reached over the web.

An attacker who knows where to look can reach something there during the minutes the update runs, and make the server execute it.

The record's title is 'Unauthenticated Upgrade-Window Staged-Installer Remote Code Execution'. Each word matters. Unauthenticated: no login. Upgrade-window: only while an upgrade is in progress. Staged-installer: the thing reached is part of the staged copy of the new software, not the live site. Remote code execution: the attacker's instructions run on your server.

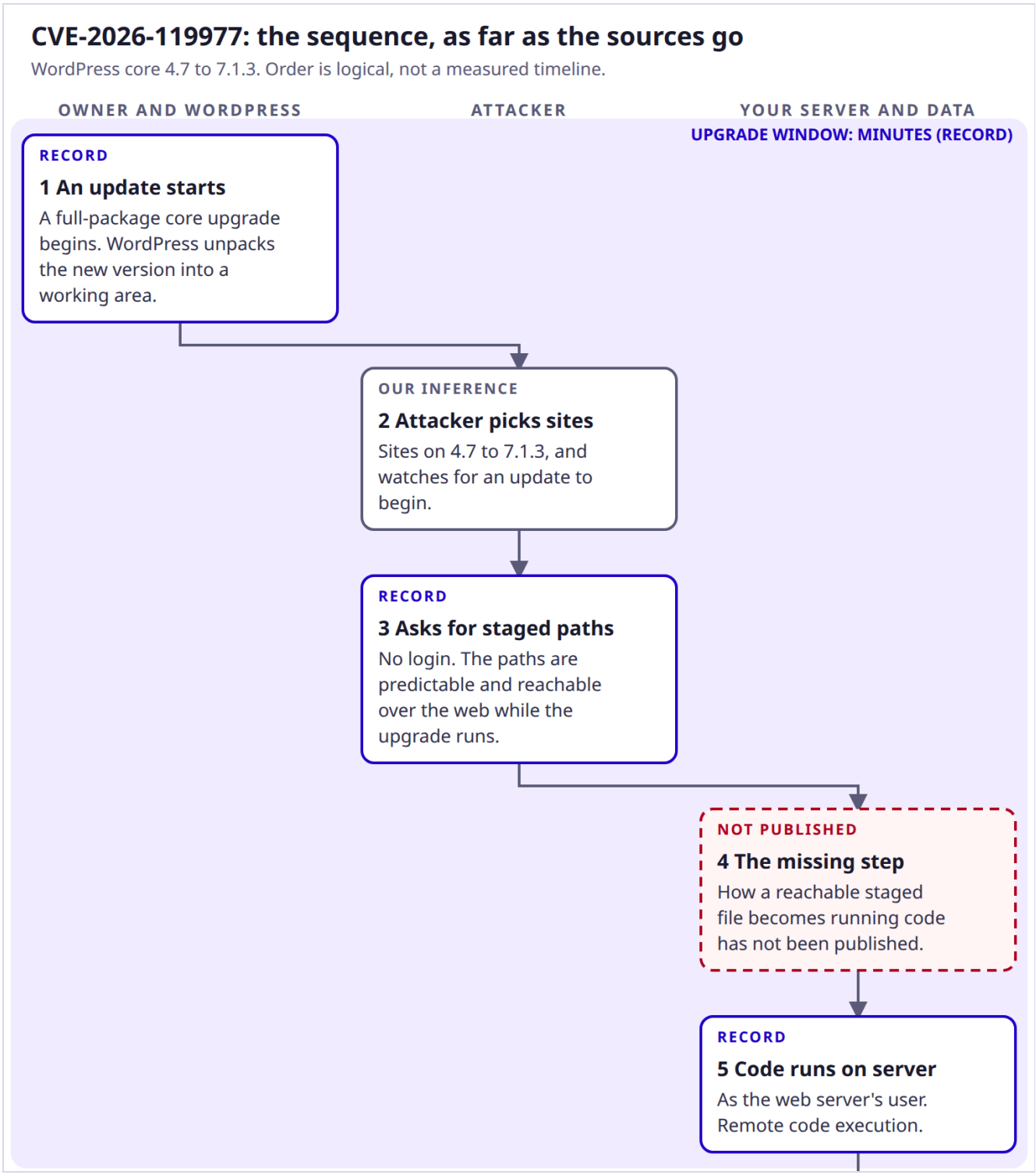
A time-of-check to time-of-use flaw is one where the software checks that something is safe, then uses it a moment later, and the thing can change in between. Here the gap is the upgrade itself.

What we have not read: the technical write-up from Wordfence (its page returned a bot challenge to us) and the source code change (the WordPress code browser refused our request). The CVE record's references point at four files in the 7.1.3 source, three of them the upgrade classes and one, setup-config.php, the installer's configuration step. We think that last reference hints at how an attacker gets from the staged files to running code, but that is our inference, and we do not describe a mechanism beyond what the record says.

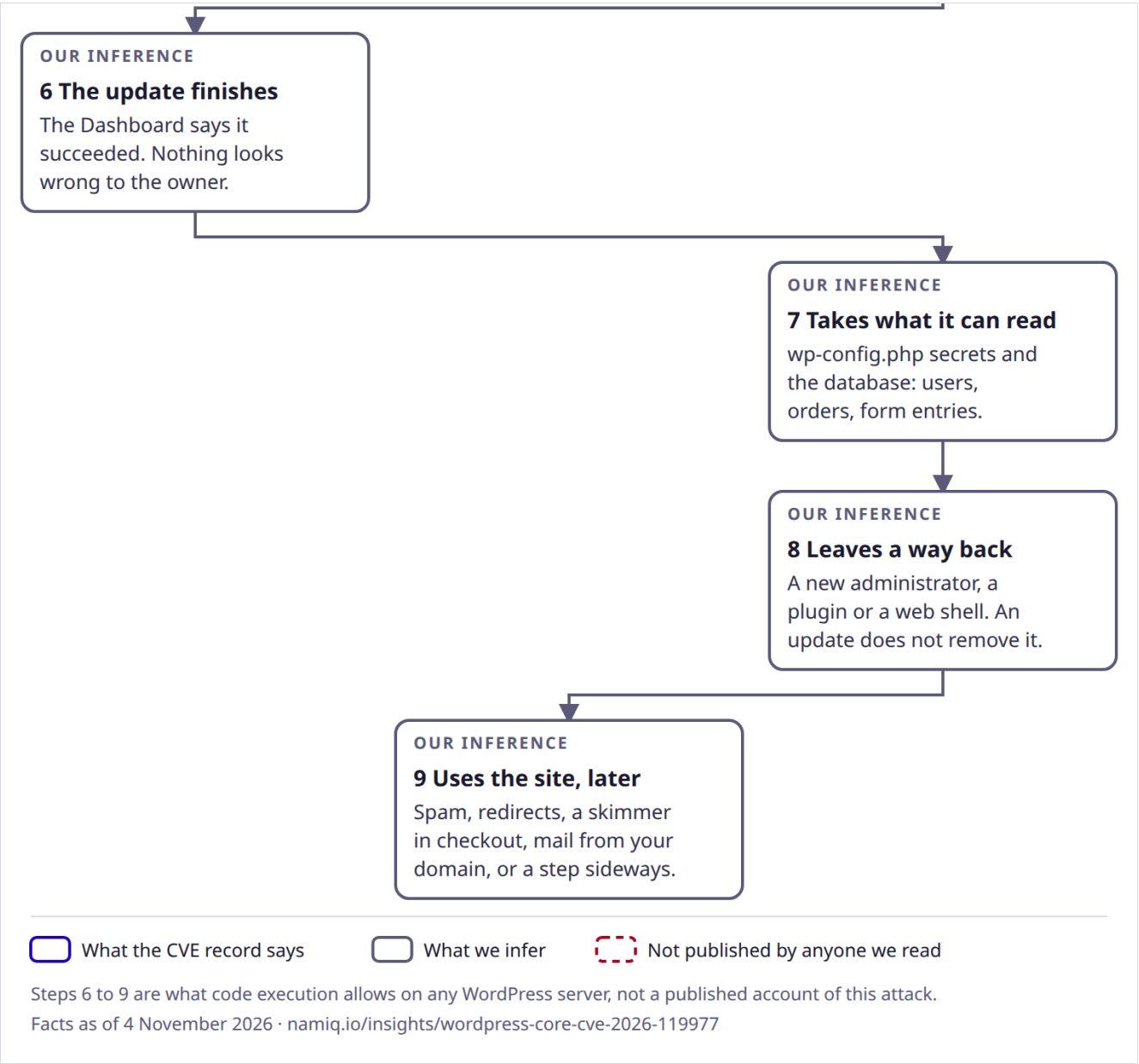
The attack, step by step

The drawing shows the sequence as far as the sources go. Every step carries one of three labels: what the CVE record says, what we infer, and what nobody we read has published. Steps 1 to 5 are the attack itself. The record supports three of them (1, 3 and 5), step 2 is our inference and step 4 is not published. Steps 6 to 9 are not a published account of this attack. They are what code execution

as the web server's user allows on any WordPress site, set in the order that makes sense for someone who wants to stay.



The attack sequence, steps 1 to 5. Facts as of 4 November 2026.



The attack sequence, steps 6 to 9. These are what code execution allows on any WordPress server, not a published account of this attack.

The same nine steps in words, each with where it comes from

Step	What happens
1. The update starts	Record: a full-package core upgrade begins and WordPress stages the new version in a working area. Which update methods open the window is not stated.
2. The attacker picks sites	Our inference: sites on 4.7 to 7.1.3 are the candidates, and an attacker who wants to hit a short window has to watch for it.
3. It asks for the staged paths	Record: the staging paths are predictable and reachable over the web, and no login is needed, during the upgrade window.

4. The missing step	Not published: how a reachable staged file is made to run. The record's title says 'staged installer'. We could not read the technical write-up or the code change.
5. Code runs on the server	Record: remote code execution, so the attacker's code runs as the web server's user.
6. The update finishes	Our inference: the window closes and the update completes. The Dashboard reports success, so the owner has no reason to look.
7. It takes what it can read	Our inference, general to code execution: the secrets in wp-config.php and the database, which holds users, orders and form entries.
8. It leaves a way back	Our inference, general to code execution: a new administrator, a plugin or must-use plugin, or a web shell. An update replaces core files and does not remove these.
9. It uses the site, later	Our inference, general to code execution: spam pages, redirects, a skimmer in the checkout, mail from your domain, or a step to other sites on the same server.

Three things follow from the shape.

- Steps 1 to 5 take minutes and leave little behind. The owner sees an ordinary update. Only the access logs and the files hold any trace.
- Step 4 decides how bad this is, and it is the one nobody has published. If a reachable staged file can be made to run with one request, every update on an exposed site is a chance. If it needs a rare condition, far fewer are. CISA's rating of 'automatable: no' fits the second reading. It does not prove it.
- Step 8 is why updating is only the first job. The fix closes the route in at step 3 to 5. It does not remove what step 8 left.

Why a 'high complexity' flaw is on the exploited list

The score says Attack Complexity High, and that is why this reads as 8.1 rather than 9.8. It is a fair description of one site: the attacker has to be there while an update runs, and an update runs for minutes. It is a poor description of the web. Many sites update on automatic schedules every day, the paths are predictable, and the request needed is cheap. Someone who watches a large number of sites at once does not need one lucky timing. They need the update windows of some of those sites to fall under their requests, and over enough sites they will.

That is our reasoning, not a published account of how the attacks work. What is published is CISA's finding that exploitation is active. The EPSS score from FIRST, read on 4 November, is 0.0102, about 1.0 per cent, at the 63.5th percentile. A score that low next to a CISA exploitation entry reads to us as a score that has not caught up with a listing made yesterday. Either way, do not let it set your priority.

The unusual part: the fix runs inside the problem

A site that is on an affected version and never updates is, on the record's description, not in an upgrade window. A site that updates is. So the act of moving to the fixed release is itself the exposed moment, and it is carried out by the old code.

We do not know how far this goes. WordPress's updater replaces some of its own files as part of the update, and the record also points at two of the same upgrade files in the 7.1.4 source, which suggests the fix touches them. Whether the 7.1.3 to 7.1.4 update runs entirely under the old behaviour, partly under the new, or not at all in the window, is a question for the vendor. The release post does not say.

What follows from that, and is our advice rather than the vendor's: shorten the window and shrink who can reach the site while it is open. Update at a quiet hour. If you can, keep outside traffic from reaching the site for the minutes the update takes, at the host's firewall, the CDN or the load balancer, and restore it when the update has finished and been verified. If you cannot, update anyway. An unpatched site that is under active attack is a bigger risk than a few minutes of window.

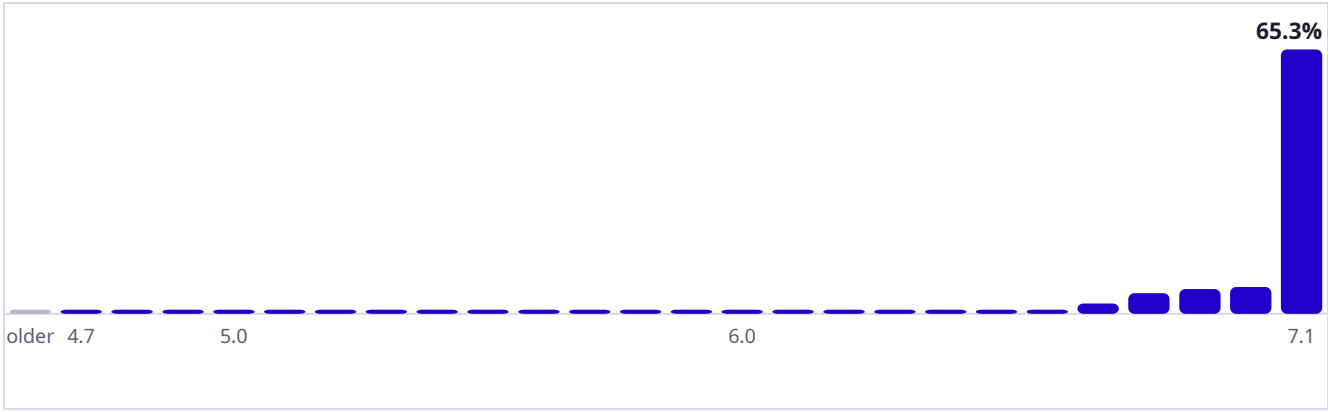
How far it reaches: the web, and your data

39.5%
of all websites run WordPress
W3Techs, page dated 5 November 2026

58.2%
of the sites whose content management system is known
W3Techs, page dated 5 November 2026

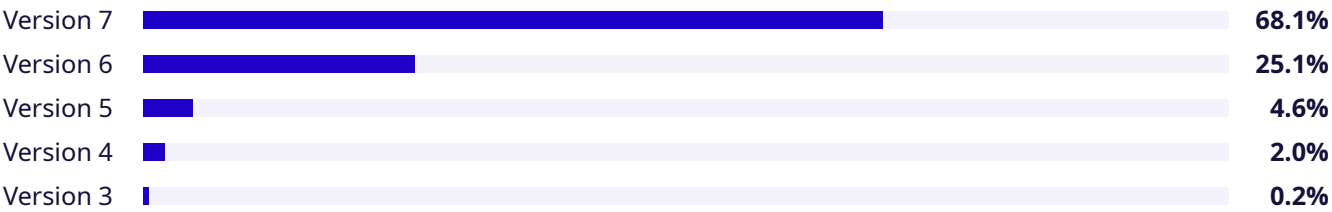
99.0%
of sites that check in with WordPress.org are on a branch the CVE names (4.7 to 7.1)
WordPress.org, read 4 November 2026

Share of checking-in sites, by WordPress branch (WordPress.org, read 4 November 2026). Dark: a branch the CVE names, 99.0% together. Grey: older than 4.7, 1.0%.



Counted by branch, not by patch, so it cannot tell 7.1.3 (affected) from 7.1.4 (fixed). It shows how wide the range is, not how many sites are still unpatched. The 7.2 beta (0.01%) is left out of the range.

WordPress sites by major version (W3Techs, page dated 5 November 2026)



Of the WordPress sites W3Techs can detect. The two sources count different sets of sites and are not added or compared: on version 7 alone, WordPress.org shows 72.0% of its checking-in sites and W3Techs shows 68.1% of the sites it detects. Neither is a count of sites.

W3Techs, read on 4 November 2026 (its page is dated 5 November 2026, UTC), reports that WordPress is used by 58.2 per cent of the websites whose content management system it can identify, which it puts at 39.5 per cent of all websites. Its version split for WordPress sites is 68.1 per cent on version 7, 25.1 on 6, 4.6 on 5, 2.0 on 4 and 0.2 on 3. These are W3Techs's measurements of sites it can detect; we did not verify them independently and we do not convert them into a count of sites.

WordPress.org publishes the share of sites that check in with it by branch. Read on 4 November 2026, 99.0 per cent are on branches from 4.7 through 7.1, which is the range this CVE names: 65.3 per cent are on 7.1, 6.7 on 7.0 and 27.0 on 4.7 through 6.9. 1.0 per cent are on older branches. Those figures are by branch, so they cannot tell 7.1.3 from 7.1.4. They show how wide the range is, not how many sites are still unpatched. Nobody has published that number.

Our reading is that nearly the whole running WordPress base sits inside the affected range at some point in its update cycle, and that the number of sites that are exposed on any given day is a smaller and unknown fraction, because the risk is in the window. That fraction is not small enough to ignore, because CISA has seen exploitation.

What code execution on a WordPress server gets an attacker

This is general to remote code execution as the web server's user, not specific to a published attack on this flaw. It is the list to put in front of an owner.

- wp-config.php: the database name, user and password, the authentication keys and salts, and any mail or API credentials kept there. With the database password, the attacker has the database without needing the site.
- The database: customers, orders, form submissions, membership lists, support tickets, user accounts and password hashes, and whatever plugins store there. For a shop that includes billing and delivery addresses. For a clinic, a law firm or a school it can include far more sensitive records.
- The files: every upload, every private document that was placed in the media library, every backup left in a web-readable folder.
- The ability to change the site: new administrator accounts, planted plugins or must-use plugins that survive updates, edited theme files, a web shell for later.
- The ability to use the site: serving spam pages or malicious redirects to visitors, skimming payment details in the checkout, sending mail from your domain, or hosting someone else's content.
- A way sideways. On shared hosting, other sites under the same account or server may be reachable. On a host that runs many clients, one compromised site can become many.

A compromise that stays quiet is worse than one that is loud. Defacement is noticed in an hour. A skimmer in a checkout, a new administrator, or a hidden plugin can run for months.

After the code runs: what happens, in time order

This is the part the score does not show. The table is general to code execution on a WordPress server and is not a published account of attacks on this flaw. The first column says when, the third says what the owner could see, and that is usually less than the second says is happening.

What follows code execution on a WordPress site, in time order. Our reading, general to the class of flaw.

When	What the attacker can do	What the owner can see	What it means for the business and its data

Minutes	Read wp-config.php and the database. Copy users, orders, form entries and anything stored in plugins. Add an administrator or a plugin that survives updates.	Nothing. The update reports success.	The data is exposed from this moment, and nobody knows. A breach-notification clock generally runs from when you become aware, so the delay is the owner's risk, not a protection.
Hours to days	Try what else the server reaches: other sites on the same account, mail credentials, backups, cloud keys stored in files.	Possibly a slow site, an unfamiliar scheduled task, or mail that bounces.	The incident stops being one site. For a host or an agency it becomes every client on the server.
Days to weeks	Put the site to use: spam pages that hurt search ranking, redirects for visitors who arrive from search, a skimmer in the checkout, mail sent from your domain.	Odd pages in search results, a browser or search warning, customers reporting redirects, a question from the payment processor.	Traffic and sales fall. For a shop, card details may be leaving with every order. The warning from a search engine or a card brand is often the first signal, and it comes from outside.
Weeks to months	Come back through the way left behind. Reuse stolen credentials on other services where people reuse passwords. Hand the access on.	A second incident after the first looked closed. Your credentials turning up elsewhere.	The cost repeats. Contracts, questionnaires and insurers ask when you patched and when you knew. The answer has to be a dated file.

Two points come out of the table. The cost of a compromise is set mostly by how long it runs unseen, not by how it began. And the first people to see it are usually not the owner: a search engine, a customer or a payment processor. A site owner who waits to be told is waiting for the slowest signal.

Where each step can be broken

Controls against each step of the sequence, and how sure we are of each. Our reading.

Step	What breaks it	How sure we are
1 to 3: the window	Update at a quiet hour, and keep outside traffic away from the site while the update runs.	Our suggestion. It follows from the record's 'during the upgrade window'. We have not tested it, and the vendor has not said it works.
2: finding the site	A site that does not have to be public sits behind a login at the edge, a VPN or an IP allow list.	Our suggestion. Strongest where the site has no reason to be open to everyone.

3: the request	A firewall rule against the staging paths.	Unproven. The record gives no path we could write a rule from, and we have seen no vendor rule.
4 and 5: execution	The fixed release: 7.1.4, or the fixed patch of your branch.	The vendor's fix. We read the release post and the record's version ranges, not the code.
7: credential theft	After any suspected access, rotate the database password and the keys and salts in wp-config.php, and the credentials stored in the database or files.	General practice. It limits what a stolen copy is worth, not whether it was taken.
8: the way back	wp core verify-checksums, plugin checksums, the administrator list, must-use plugins, and PHP files changed since 28 October. Rebuild from a known good copy where the data justifies it.	Our suggestion. It finds the common leftovers, not all of them. A clean result is not proof.
6 and 9: the use	Watch from outside the site: search warnings, the integrity of the checkout page, outbound mail volume, and what your payment processor tells you.	General practice. It is how most owners would first learn.

What it costs a business, without a figure

We do not quote a cost per breach. Published averages differ by source, by country and by what they count, and a number from someone else's survey is not a number about your site. The shape of the cost is the same everywhere, and each item is something you can ask your provider about today.

- Time to find out. Before you can say what was taken you have to establish what the attacker could reach. That is skilled hours, usually from outside.
- Downtime and rebuild. A site that cannot be trusted is rebuilt from a known good copy and re-checked. A shop that is offline sells nothing for the duration.
- Search and browser penalties. If the site serves malware or spam, search engines and browsers can flag it, and the traffic goes away before the site does.
- Notification. If personal data was taken, a law and often a contract says who must be told and by when. In the European Union the GDPR (Article 33) sets a 72-hour limit for telling the regulator once you are aware of a breach, where that is feasible. US state laws and sector rules such as health privacy law set their own clocks. This is general background, not legal advice, and the clock runs from when you become aware of a breach, not from the day this advisory appeared. Ask your counsel which applies.
- Payment-card consequences. A shop that handles cards through a hosted payment page is in a better position than one that takes them on its own pages. A skimmer in a checkout is a card-brand matter as well as a site matter.
- Customers and partners. A client's security questionnaire can ask whether you patched known exploited flaws promptly. That is easy to answer when the answer is dated and filed.

- Insurance and contracts. Some policies and some customer contracts depend on timely patching. Read yours.

Who has the most to lose

- Agencies and managed hosts. One decision becomes hundreds of sites. They have the greatest exposure and the best chance of acting well, because they can inventory, order and verify.
- Online shops and membership sites. They hold the data that triggers notification duties and the checkout that skimmers target.
- Regulated organisations with a WordPress front end: clinics, law and accountancy firms, schools, charities. The site may hold little, but a foothold on the server next to something that holds more is the risk.
- Publishers and media. Their risk is trust and traffic more than data.
- Small businesses with a site built years ago by someone who has moved on. They are the most likely to be on an old branch, on a host that does not manage updates, and without anyone who would notice.

Things people will get wrong

- "We are on a managed host, so it is handled." Maybe. Ask when the site was patched and whether it was checked first. A patch date after 28 October with no check is not the same as clean.
- "Our site does not update, so it cannot be hit." On the record's description, an idle site is not in the window. It is also still on a version with a known exploited flaw and with eleven other security fixes outstanding, and the day it does update is the day it is exposed. Do not rely on idleness.
- "A security plugin or firewall covers this." We have not tested it and the record does not say. A rule written against paths that are predictable may help. Treat it as a second layer, not the fix.
- "It needs perfect timing, so it is rare." CISA records active exploitation. Rarity per site is not rarity across the web.
- "Version 7.1.4 is the only fix." Not if you are on an older branch. The project has issued a fixed patch on each, and the update check will offer it. Moving a large site to a new major version as an emergency is its own risk.
- "Updating was the last step." It is the first of two. The second is the question of whether anyone used the window before you closed it.
- "Versions before 4.7 are not listed, so they are fine." The record says nothing about them. They have had no fix issued for years. Treat unlisted as unknown.

How we would triage it

This is our judgement, built on the facts above.

- Find every WordPress you are responsible for, including the forgotten ones: old campaign sites, staging copies, a blog on a subdomain, a microsite built by a former agency. An unknown site is a site that is not patched.
- Rank by what an attacker would reach: data first, then whether it takes money, then whether it is reachable from the internet, then size.
- Copy before you change. For a site you will check for compromise, take a copy of the files and the database and keep the logs, so that an update does not overwrite what you are about to look for.
- Update the highest-ranked sites first, at a quiet hour, with outside traffic kept away where you can for the minutes it takes.
- Verify and check, then record the date.

Step 1: find the version and how it updates

Run these from the WordPress directory as the site's user. WP-CLI is a command-line tool many hosts provide. If you do not have it, the version is on the Dashboard under Updates, and the file wp-includes/version.php holds it.

```
wp core version
wp core check-update

# Is automatic background updating switched on, and for what?
wp config get WP_AUTO_UPDATE_CORE 2>/dev/null
wp plugin list --status=active --fields=name,version,update
```

A site that returns nothing from the config line is using WordPress's default, which is to apply minor and security releases on its own.

Step 2: update to your branch's fixed release

```
# The update check will offer the latest patch of your branch
wp core update
wp core version      # expect 7.1.4, or the fixed patch of your branch

# If you cannot use the command line: Dashboard > Updates > Update Now
```

Do this at a quiet hour. If you control the edge, keep outside traffic from the site for the duration and restore it when the update has finished. Take a backup first if you do not already have one. If an update fails part-way, do not leave the site half-updated: the release post points to a HelpHub page for 7.1.4, and your host's support is who to call.

Step 3: look for someone who got there first

The commands below only read. They look for the footprints any code execution on a WordPress site tends to leave. They are our suggestions and not published indicators of this attack, which the sources do not provide.

```
# Core files that differ from the official release, or that should not be there
wp core verify-checksums

# Plugins from wordpress.org that differ from the official release
wp plugin verify-checksums --all

# Administrators: anyone you do not know, or registered since 28 October
wp user list --role=administrator --fields=ID,user_login,user_email,user_registered

# Where the site thinks it lives (a hijack often changes these)
wp option get siteurl
wp option get home

# PHP files changed since the disclosure date, and PHP files inside uploads
find . -name '*.php' -newermt '2026-10-28' -not -path './wp-content/cache/*' 2>/dev/null |
head -100
find wp-content/uploads -name '*.php' 2>/dev/null
ls -la wp-content/mu-plugins wp-content/upgrade .maintenance 2>/dev/null
```

The last line lists must-use plugins, which load on every request and do not appear in the normal plugin list, the upgrade working folder, and the maintenance marker WordPress leaves while an update runs. We do not know that the upgrade folder is the path the flaw uses, so a leftover there is a reason to look, not a finding.

```
# Web server access logs: requests aimed at the installer or the upgrade area
grep -E 'setup-config\.php|/wp-content/upgrade/' /var/log/nginx/access.log*
/var/log/apache2/access.log* 2>/dev/null | head -50
```

The log paths are examples; use the ones your server writes. A hit does not by itself mean the attack worked, and no hit does not mean it did not: logs rotate, and exploitation may have begun before 28 October. The sources do not say when it began. A file you cannot explain in a place PHP can run is the stronger signal.

Step 4: if you find something

- Copy the files, the database and the logs, then isolate the site from the network. Do not delete first.
- Assume what the web server user could read: wp-config.php with the database password and keys, the database, and any API keys in either. Rotate them. Changing the authentication salts logs everyone out, and that is wanted.

- Reset every administrator password and remove accounts nobody can explain.
- Decide whether to clean or rebuild. A web shell is the part you found, not necessarily the only thing placed. If the site holds anything that matters, we would rebuild from a known good copy and restore content, not code.
- Tell the person who handles breach notification the same day, with the dates you found. The clock may already be running.

The first-15-seconds page is a one-page plan for the first minutes, and it applies here.

Detecting it: four events, and which ones prove anything

Step 4 of the sequence is not published, so nobody can hand you an exact signature for the request that starts it, and we do not pretend to. The chain below does not need one. It follows the drawing, and for each event it says where to look, what it can show and what it cannot. It is our design for an analyst, or for a script you control and review. Nothing in it decides for you.

The detection chain, one event per stage of the attack

Step	What happens
Event 1: a request to the staging or installer area during an update	Source: the web server access log. A request, from a client with no session, to a path under the web root that exists only while an update runs, inside the minutes the update runs. Weak alone: scanners ask for installer paths all day, and a real visitor can reach the site during an update. We do not know the exact paths.
Event 2: a PHP file appears or changes where PHP can run	Source: file integrity monitoring, or a find. A new or changed .php file in the staging area, in uploads, or in a core folder, inside or just after the window. Stronger, but an ordinary update writes many files, so compare against the release's own checksums.
Event 3: the web server starts something it normally does not	Source: process telemetry, for example the Linux audit log or an endpoint agent. The PHP or web server process starting a shell, curl, wget, nc or an interpreter. Strong when it happens. PHP can read files, query the database and make connections without starting any process, so its absence proves little.
Event 4: what the intruder does with the foothold	Sources: the account list, the database, file changes and outbound connections. A new administrator, a changed siteurl or home, a new must-use plugin, or the web server's user connecting to a destination it has never used.

How to join the four events

- Fix the window first. For each site, record when an update starts and ends: the maintenance marker (.maintenance) appearing in and leaving the web root, the email WordPress sends after an automatic update, and your host's job log. Without a window, event 1 has nothing to be compared with. We have not tested that the marker is present for the whole window.

- Join 1 to 2 by time and client. A file change within a few minutes of a request from one address, inside the window.
- Join 2 to 3 by path. The process event should name, or run from, the file that appeared. The audit log records the executable and the working directory.
- Join 3 to 4 by time. An outbound connection, a new account or a changed option soon after the process event.
- Make the clocks agree. Web, file and process logs need one time source. A join that fails by a few seconds loses the one case that mattered.

How much each combination proves. Our reading; the thresholds are judgement, not measurement.

What you see	Reading	What a person does
Event 1 only	Suspicious, not a compromise. Scanners and ordinary visitors both produce it.	Keep the log lines. Check the files for that window.
Events 1 and 2, inside one window	Probable. A changed file the update cannot explain, soon after an unauthenticated request.	Preserve evidence. Examine the file. Treat the site as under investigation.
Events 1, 2 and 3	Treat as a compromise.	Contain now: take the site away from the internet, copy the evidence, rotate secrets.
Event 4 alone	High suspicion of compromise, from some route. Each sign can have an ordinary cause: an administrator someone forgot, a planned move of the site, a new mail relay.	Confirm each sign is expected. If one is not, treat it as an incident and find how: a plugin, a stolen password, this flaw. Do not assume it was this one.
No events	Not proof of safety. Logs rotate, PHP runs without starting a process, and step 4 is not published.	Do the file and account checks anyway.

Building it: the rules and the searches

These are starting points to adapt, not tested products. Run them on a copy first and tune them against what your site does normally. Several plugins start programs for good reasons, such as image conversion and scheduled jobs, and they will match.

```
# Event 3, Linux auditd: log programs the web user starts.
# Replace 33 with the user id: id -u www-data
-a always,exit -F arch=b64 -S execve -F euid=33 -k wp-webuser-exec

# Read it back
ausearch -k wp-webuser-exec -ts today | aureport -x --summary
```

```
# Event 3, Sigma rule: process_creation, Linux
title: Web server or PHP process starts a shell or download tool
status: experimental
logsource:
  category: process_creation
  product: linux
detection:
  parent:
    ParentImage|contains:
      - 'php-fpm'
      - 'php-cgi'
      - 'lsphp'
      - 'apache2'
      - 'httpd'
  child:
    Image|endswith:
      - '/sh'
      - '/bash'
      - '/dash'
      - '/curl'
      - '/wget'
      - '/nc'
      - '/ncat'
      - '/python3'
      - '/perl'
  condition: parent and child
falsepositives:
  - Plugins that call programs on purpose; scheduled jobs; deployment scripts
level: medium
```

```
# Event 4, egress: what is the web server's user connected to right now?
ss -tnp 2>/dev/null | grep -E 'php|apache2|httpd'

# Keep a short list of destinations that user should reach
# (mail relay, update server, payment provider). Review the rest.
```

To learn the staging paths on your own estate rather than wait for a published list, run a test update on a copy of one site with access logging on. In a second terminal, watch the web root while it runs, then keep the list of what appeared and disappeared. That list is your event 1 and event 2 rule, written for your layout.

```
# On a COPY of the site only, during a test update:
watch -n 1 'ls -la .maintenance wp-content/upgrade 2>/dev/null'
```

What the chain maps to in ATT&CK

The mapping is ours. MITRE and CISA have not published one for this CVE, so read each line as 'likely'. Event 1 and the access that follows it: T1190 Exploit Public-Facing Application. Event 2 and a

way back left in step 8: T1505.003 Web Shell. Event 3: T1059.004 Unix Shell. Event 4, a new administrator: T1136.001 Local Account.

What stays with a person

A script can take the reversible steps when the table says to contain: block the site at the edge, take a snapshot, copy the logs. We would keep four decisions with a person: whether a medium-confidence case is a compromise, whether to rebuild or clean, whether anything must be notified, and when the site comes back. Each of them depends on what the site holds and who it belongs to, and none of them can be undone by a second run.

What this cannot do

- It cannot give you the request pattern for event 1. That needs the technical write-up or a test on your own layout.
- It only sees what you log. Process and file monitoring are off on most small sites, and turning them on after an incident does not recover the window you missed.
- Without a process event, it leans on files, accounts and egress. That is a weaker position, and the table says so.
- It does not prove the flaw was used. A match is a lead for an analyst, not a finding.

A plan for agencies and hosts with many sites

- Inventory in one command per server. Core version, update method and whether the site is public, written to one table. Run it today, not after the update.
- Sort into three lists: data-bearing and public, public with little data, internal. Work them in that order.
- Update in small batches at quiet hours. Many sites updating together makes a long window for everyone in it. A batch of ten you can verify is better than a hundred you cannot.
- Verify each site with the same checks and keep the output. A client who later asks 'were we checked?' deserves a dated file, not a recollection.
- Tell clients. A short message with what was exposed, what you did and when, and what you found. Silence reads as not knowing.
- Fix the process that left you with unknown sites. Every forgotten site on the first list is a finding.

If you cannot update this week

- Keep the site from reachable-by-everyone: put it behind a login at the edge, a VPN or an IP allow list until you can update. This is the strongest option we can offer, and it is our

suggestion, not the vendor's.

- Turn off automatic core updates only if you will update by hand within days. Switching off an update to avoid the window leaves the flaw and the eleven other security fixes in place. It is a bad trade unless it is short.
- Put a web application firewall in front, and read its vendor's note on this CVE. We have not seen one and do not claim it works.
- Take a fresh backup, offline, and test that it restores.

What CISA now asks for

CISA's required action on the entry is to apply the vendor's mitigations under its binding directive BOD 26-04 and to follow its forensics triage requirements. Its entry also records 'forensic triage: Yes'. That the second step is on the entry at all is the signal: for a flaw already used on real sites, the question includes whether anyone used it before you patched. The directive binds US federal civilian agencies, not a private company, and the due date is theirs. As a private owner, you can still use it as the date that other people will measure you against. It is 6 November, two days after the date of this note.

What to send your provider today

> *We run WordPress sites on your service. CISA added CVE-2026-119977 (WordPress core 4.7 to 7.1.3, upgrade-window remote code execution) to its exploited list on 3 November, due 6 November. Please tell us today, for each of our sites: the core version now and the date it reached the fixed release; whether the update ran at a time when outside traffic could reach the site; and whether you checked for a compromise before the update (core checksums, new administrators, changed PHP files, requests to the installer or upgrade paths in the logs) and what you found. If you cannot answer for a site, tell us which and when you will.*

What we do not know yet

- Step 4 of the sequence: how a reachable staged file is made to run.
- The request pattern for event 1 of the detection chain. We have no published path to write it from.
- How the exploitation works and when it started. The CVE record is three sentences, the vendor's post is one line, and the technical write-up named by CISA was not readable to us.
- Whether the update from 7.1.3 to 7.1.4 itself runs inside the window. The release post does not say.
- Which update methods are exposed: automatic background updates, the Dashboard, WP-CLI, Composer-managed installs, rebuilt container images. The record says 'full-package core upgrades' and no more.

- How many sites remain unpatched. The published statistics are by branch, not by patch.
- Who is behind the exploitation. The sources name no one.
- Whether a firewall, security plugin or edge rule stops the attack. We have not tested one.
- Whether versions before 4.7 are affected. The record is silent.

If you take one thing from this: WordPress's own update is both the fix and the moment of risk. Make the window short, keep outsiders away from it where you can, and treat the update as the first of two jobs, with the check for a prior compromise as the second.

Appendix 1. Severity decoded

CVE-2026-119977: score 8.1, `CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H` The score is held down by Attack Complexity High, which reflects the timing: the attacker has to act while an upgrade is running. Nothing else in the vector is mild. No account, no click and no special network position are needed, and all three of confidentiality, integrity and availability are total.

Metric	Value	What it means here
Attack vector	Network	Reachable over a network, with no need to be on the same segment.
Attack complexity	High	Needs conditions outside the attacker's control.
Privileges required	None	No account is needed.
User interaction	None	No one has to do anything.
Scope	Unchanged	Damage stays inside the vulnerable component.
Confidentiality	High	Total loss of confidentiality.
Integrity	High	Total loss of integrity.
Availability	High	Total loss of availability.

Appendix 2. Timeline

Date	Event
28 October 2026	The CVE record's own timeline lists this date as 'Disclosed'.
30 October 2026	The CVE identifier was reserved (17:17 UTC), two days after the 'Disclosed' date in the same record. We do not know why the order is this way.
31 October 2026	Record published by the CNA (Wordfence), 05:33 UTC.
October 2026	WordPress 7.1.4 security release announced. The post's address is dated 2026/10; we did not read a day. Backports to older branches 'in progress' when the post was written.
3 November 2026	CISA adds the flaw to its Known Exploited Vulnerabilities catalogue, with a due date of 6 November. Its coordinator rates exploitation 'active'.
4 November 2026	Record updated (04:56 UTC). WordPress.org's update check lists a fixed release on each of the 25 affected branches (4.7.39 through 7.0.8, and 7.1.4).
6 November 2026	CISA due date. Two days after the date of this note.

Appendix 3. Exposure by situation

These priorities are our judgement from the attacker prerequisites in the sources. They are not vendor ratings.

Your situation	Priority	Timeframe
Online shop, membership or booking site on WordPress 4.7 to 7.1.3, reachable from the internet	Critical	Update today, then check for a prior compromise today
Agency, host or managed provider running many WordPress sites	Critical	Start the fleet plan today; highest-data sites first
Any internet-facing WordPress that stores customer, patient, client or staff data	Critical	Update today; involve whoever handles breach notification if the check finds anything
Brochure or marketing site with a contact form, on automatic core updates, reachable from the internet	High	Update before CISA's 6 November date; check
WordPress updated only by hand, WP-CLI, Composer or a rebuilt container image	High	Update this week; ask the vendor whether your method opens the same window (we could not confirm)
WordPress reachable only from an internal network or behind a VPN	Medium	Update this week; anyone on that network is a possible attacker
WordPress older than 4.7	High	Plan a move to a supported release. The record does not list these versions as affected and does not clear them; no fix is published for them
Already on 7.1.4 or your branch's backport, and checked clean	Low	Keep the check dated and filed
Managed WordPress host that says it has patched	Medium	Ask when, and whether they checked for a compromise before patching

Appendix 4. Definition of done

- Every WordPress site you own or host is on a list with its core version, how it updates, and whether it is reachable from the internet.
- Each site in the affected range is on its branch's fixed release (7.1.4, or 7.0.8, 6.9.11 and so on), confirmed with wp core version or the Dashboard, not assumed.
- wp core verify-checksums reports no changed or unexpected core file on each site, or each finding has been explained.

- The administrator list matches the people you know; no account was added since 28 October that nobody can explain.
- Access logs back to 28 October (further if you keep them) have been searched around each update and each automatic update, and the result is written down with the date.
- Your host or agency has said in writing when each site was patched and whether it was checked before the patch.
- If anything was found: the site was isolated, evidence copied, secrets in wp-config.php and anything stored in the database rotated, and the person who handles breach notification has been told.
- A date is set to re-read the WordPress and CISA entries, because the vendor has not published a full technical account.

Sources

- CVE-2026-119977 record (CNA: Wordfence), with CISA ADP data: CVSS, CWE-367, affected ranges, references, timeline: https://www.cve.org/CVERecord?id=CVE-2026-119977
- CISA Known Exploited Vulnerabilities catalog, entry for CVE-2026-119977 (added 3 November 2026, due 6 November): https://www.cisa.gov/known-exploited-vulnerabilities-catalog?field_cve=CVE-2026-119977
- WordPress 7.1.4 release post: the thirteen items, the backport statement, credit to the reporter: https://wordpress.org/news/2026/10/wordpress-7-1-4/
- CISA BOD 26-04: Prioritizing Security Updates Based on Risk: https://www.cisa.gov/news-events/directives/bod-26-04-prioritizing-security-updates-based-risk
- MITRE ATT&CK: T1190 Exploit Public-Facing Application, T1505.003 Web Shell, T1136.001 Local Account, T1059.004 Unix Shell. ATT&CK is a registered trademark of The MITRE Corporation. Technique names are reproduced and distributed with the permission of The MITRE Corporation under the ATT&CK Terms of Use.: https://attack.mitre.org/resources/terms-of-use/
- Sigma rule format specification (for the process rule above): https://github.com/SigmaHQ/sigma-specification
- FIRST EPSS, CVE-2026-119977 (EPSS by FIRST and Empirical Security; read 4 November 2026): https://api.first.org/data/v1/epss?cve=CVE-2026-119977
- WordPress.org version statistics (share of checking-in sites by branch, read 4 November 2026): https://api.wordpress.org/stats/wordpress/1.0/
- WordPress.org core stable-check (which releases it flags as latest, outdated or insecure, read 4 November 2026): https://api.wordpress.org/core/stable-check/1.0/
- W3Techs: WordPress usage statistics and market share (read 4 November 2026): https://w3techs.com/technologies/details/cm-wordpress